1 WHAT IS AGILE?

1.1 THE HISTORY OF AGILE

‘Standing on the shoulders of giants’ is a particularly apt term when discussing the evolution of Agile, as Agile thinking is founded on the concepts and ideas behind many different IT governance and delivery frameworks. Table 1.1 shows the evolution of these frameworks (see Chapter 14) since the late 1940s, leading to what is now generically referred to as ‘Agile’.

Table 1.1 History of Agile frameworks

Year	Development of framework
1948	Taiichi Ohno, Shigeo Shingo and Eiji Toyoda create the 'Toyota Way'. Many Agile concepts relate to Lean thinking
1985	Tom Gilb develops EVO (Gilb, n.d.)
1986	Barry Boehm works on Spiral (Boehm, 1986)
1990	Rapid Application Development (RAD) is documented (Martin J., n.d.)
1992	The Crystal family of methodologies is defined (Cockburn, 2004)
1994	Dynamic Systems Development Method is created (DSDM Consortium, 2014b)
1995	Ken Schwaber and Jeff Sutherland present a paper on Scrum at the OOPSLA (Object-Oriented Programming, Systems, Languages and Applications conference) (Sutherland, Patel, Casanave, Miller and Hollowell, 1995)
1996	Rational Unified Process (RUP) (IBMRational, n.d.)
1997	Feature Driven Development
1999	Kent Beck publishes *Extreme Programming (XP) Explained* (Beck, 2004)
2001	The Agile Manifesto is created (Agile Manifesto, 2001)
2003	Mary and Tom Poppendieck publish Lean Software Development (Poppendieck, 2003)
2007	David J. Anderson discusses Kanban at Agile 2007 (Anderson, 2010)
2009	Eric Ries speaks on Lean start-up (Ries, 2011)

Some Agile thinking is based on the 'Toyota Production System (TPS)' (Liker, 2004) or 'Lean', as it is now more widely known. For example, the concept of Visual Boards (see Section 8.7) is taken from TPS. However, as the name suggests, TPS is mainly related to manufacturing products on a production line. Agile thinking is more focused on Lean product development than it is on Lean manufacturing, because the dynamic environments in which Agile is implemented are more similar to a dynamic innovative Lean product environment than a Lean manufacturing environment where variability is specifically discouraged and repeatability encouraged. Innovation and creativity, two fundamentals of effective product development, are enabled by variability and disabled by focusing on repeatability.

Tom Gilb's 'Evo' and Barry Boehm's 'Spiral' approaches were incorporated into Agile thinking in the early 1990s into what became known as Rapid Application Development (RAD).

Many RAD frameworks up to this point had concentrated on product delivery with no great focus on project governance. The exception was DSDM (see Section 14.3), which was created in the mid-1990s and focused on delivery within projects.

In 1999, another key Agile framework was created called eXtreme Programming or 'XP' (see Section 14.1). While XP focuses more on the values and practices associated with the technical programming aspects of engineering software, many of the practices are now also being used effectively in generic product development.

An important point in the evolution of these frameworks occurred when the term 'RAD' became associated with delivery failure, and finally disappeared in the late 1990s. One of the reasons for this was that, in the late 1990s, many organisations were using the term RAD to describe their delivery method, whether it was actually RAD or not, because it had become the 'cool' name in town. This meant that teams and organisations pretended or imagined that they were doing RAD, but did it without actually changing any of their delivery and management culture and behaviours. This led to a lot of failed initiatives that were shaped in a traditional 'Waterfall' way (see Section 2.6.3) being described as 'failures of RAD' – even though they hadn't initially been properly set up as RAD projects. Fundamentally, a key point was misunderstood by many people: while RAD itself was easy, transforming to RAD could be extremely complex.

Also, the word 'Rapid' in 'Rapid Application Development' meant that the immediate and lasting impression was of speed rather than a balance of regular value delivery and quality, which proved inappropriate.

It was not until 2001, when the Agile Manifesto (see Section 1.2.1) was formulated, that a collective generic name and terms of reference that supported all the frameworks was defined, and Agile as a concept was born.

In the same year the first Scrum (see Section 14.2) book was authored by Ken Schwaber and Mike Beedle (Schwaber and Beedle, 2001). The book evolved the basic concepts of Scrum from a seminal paper in the 1986 *Harvard Business Review*, written by the godfathers of the Scrum Agile Process, Takeuchi and Nonaka (Takeuchi and Nonaka, 1986). Scrum has since grown into by far the most implemented Agile framework in the world.

Since that time, further evolution of Agile thinking has been achieved, and notable more recent frameworks include Lean software development (see Section 14.6) and Kanban (see Section 14.5), amongst others.

1.2 THE AGILE MANIFESTO

As mentioned in the Introduction, the Agile Manifesto provides the single definition of Agile and underlies the development and delivery of Agile frameworks. While its title 'The Manifesto for Agile Software Development'

suggests that it is only applicable to software development, the values and principles described in the Manifesto can easily be applied to the development of many types of product.

The Manifesto describes 4 values and 12 supporting principles. The values set out in the Agile Manifesto are:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions *over* processes and tools.

Working software *over* comprehensive documentation.

Customer collaboration *over* contract negotiation.

Responding to change *over* following a plan.

That is, while there is value in the items on the right, we value the items on the left more.

Typically an Agile audit or health check will be performed against the 4 manifesto statements and the 12 manifesto principles. It may also be further refined by the key values and principles of whatever combination of Agile frameworks (see Chapter 14) is being implemented.

In this chapter, we will take a closer look at the manifesto's values and supporting principles.

1.2.1 Agile values

The four Agile Manifesto statements or values are described next and are further expanded in Chapters 9–12.

1.2.1.1 Individuals and interactions over processes and tools

While processes and tools provide significant value to software development teams and enable them to be Agile, the best processes and tools will not help them to deliver value to the customer without enabled and motivated people who interact effectively as a team (see Chapter 9).

1.2.1.2 Working software over comprehensive documentation

The vast majority of software products require supporting documentation; for example most software deliveries require technical user documentation (see Section 8.6) as without this documentation it will become extremely difficult to support and maintain the software product in the future and throughout its lifecycle. (See also Chapter 10.)

However, while fit-for-purpose documentation is key in an Agile delivery, the focus is on providing working software, and therefore adding value directly to the customer. This means that Agile demonstrates progress through regular visible incremental deliveries of a consistently working product (see Section 10.3). Each and every time a new increment of the product is delivered the customer can be assured that good, agreed progress is being made – and/or be aware in advance of

any hindrances to progress. In Agile product development the documentation is kept as Lean as possible (see Section 8.6); only documentation that adds value to the stakeholders is produced and the production of the documentation is synchronised with the incremental delivery of the working product.

1.2.1.3 Customer collaboration over contract negotiation

In Agile it is important to create a consistently collaborative and open relationship between the customer and supplier (see Chapter 11 and Section 11.1), and to ensure that the customer and supplier acknowledge that an effective product cannot be developed without that collaboration. Obviously there are still contracts between customers and suppliers, however, contracts tend to focus on clarifying the collaboration and the working approach between the customer and supplier to ensure they can work together to a mutually satisfactory conclusion. This means that the Agile contract concentrates on enabling inspection and adaptation of the product, prioritisation and collaboration between all stakeholders.

This stands in contrast to a traditional 'Waterfall'-driven contract, in which the analysis and design stages will produce detailed documents that become fundamental parts of the contract (e.g. the requirement specification, functional specification etc.). The interaction between customer and supplier then becomes a negotiation based on the detailed documents that have been produced. This can create significant friction; for example, it may lead to each party trying to get the other to pay for changes to the contracted specifications. This can become a very significant problem where there is a large amount of change required to the contracted specifications.

1.2.1.4 Responding to change over following a plan

According to About.com, German military strategist Helmuth von Moltke wrote:

> 'No battle plan survives contact with the enemy.' As a result, he sought to maximise his chances of success by remaining flexible and ensuring that the transportation and logistical networks were in place to allow him to bring decisive force to the key points on the battlefield.
>
> (Hickman, 2014)

This principle is replicated in an Agile development: while there is a significant amount of focused planning (see Section 7.3; Chapter 12), this planning is designed to enable inspection and adaptation.

In essence the majority of Agile frameworks align to the following concepts:

- If programme/project plans are required, they are defined at a high level – these are baseline plans that are expected to change.
- If stage (or release) plans are required, they are defined at a mid level – again, these are baseline plans that are expected to change.
- Detailed work package or sprint/iteration plans (see Section 12.1) are commitment plans, the commitment being against an agreed goal (for example, an agreed increment of the product).

This means that up-front project plans and stage plans are not commitment plans; rather it is understood that these plans are forecast best guesses based on experience or factual evidence and change is anticipated as the project develops (see Section 7.3).

1.2.2 The Agile principles

There are 12 Agile principles that support the 4 manifesto statements.

1.2.2.1 Our highest priority is to satisfy the customer through early and continuous delivery of valuable software (see Section 10.1)

The entire focus of Agile is to deliver value to the customer as frequently, consistently and regularly as possible. By continuously delivering product increments that provide additional value, the customer is much more likely to be satisfied and involved. It also means that they are more likely to understand and buy into the product being delivered.

This means that in an Agile delivery, it is essential that product stories (the required features of the product – see Section 7.1) are expressed in a way that makes sense to everyone in the team, the technical people and business people. Generally the order of the stories to be delivered will be largely driven by business value; however, the team also need to ensure that the optimum way to deliver the product from a technical perspective is considered (amongst many other considerations).

1.2.2.2 Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage (see Section 12.1)

In an Agile delivery, the team and customer endeavour to refine and break down complex business needs into stories (features or components of the product – see Section 7.1) that can be understood, defined, tested and delivered fast. This approach enables the entire team to inspect, learn and adapt as they iteratively deliver product features via stories.

As the team continuously delivers value-add stories to the customer, they can also harness and demonstrate change to the customer's competitive advantage. As they collaborate effectively and reciprocally with the customer, they can identify and learn/evolve more effective ways to deliver added value (see Section 10.1).

Agile also significantly reduces the risk of the team spending a significant amount of time producing detailed specifications that might be appropriate at the time of writing them, but might not actually add much value on the day of delivery. The window of opportunity for new systems is ever more dynamic and in many cases quite small or short, and so accurate valuable delivery is paramount.

1.2.2.3 Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale (see Section 10.2)

All Agile frameworks focus on delivering value-add product increments frequently, thereby enabling fast feedback-cycles and the ability to change direction if required. This means that Agile teams should deliver value-add stories to their

customer every few weeks. Even when Agile is being scaled (see Section 14.8), and it may not be practical or feasible to deliver within a few weeks, teams should still try to deliver products in as short a period as possible – and never more than every few months.

A key metric that is therefore often implemented is 'cost of delay' (i.e. what is the cost to the business of delaying delivery of a product feature – see Section 10.1). This metric allows the customer to identify the opportunity cost of stories – meaning that they will be able to understand the relative cost of stories being available now, sooner or later.

1.2.2.4 Business people and developers must work together daily throughout the project (see Section 11.1)

In dynamic environments stories that add value and their relative priority will change as understanding evolves and business needs change. This constant 'reshuffling' requires close collaboration between the customers, stakeholders and team, as well as a common language, free from jargon. Face-to-face communication as well as a willingness to work together closely are key to creating this collaborative environment (see Section 8.2).

1.2.2.5 Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done (see Section 9.1)

It is common sense that motivated people are going to be more productive (see Section 9.1). If an organisation treats people like robots who cannot be trusted, then the people will act like robots that cannot be trusted. If the organisation treats individuals like trusted adult professionals, then they will act in that way (see Section 9.1.2).

Agile can be extremely difficult if there is a blame culture, and/or when individuals and teams are not empowered. Agile transformation is an ongoing journey. Human beings will not transform unless they understand why they are transforming and what problems they are aiming to solve by implementing Agile.

1.2.2.6 The most efficient and effective method of conveying information to and within a development team is face-to-face communication (see Section 8.2)

Communication is not solely the spoken word – a significant amount comes from visual cues and body language. Therefore, written communication is prone to misunderstanding, and while documents and emails are a great way of broadcasting information, they are not a good communication tool.

If face-to-face communication is removed, feedback cycles can become unnecessarily and dangerously extended. Fast, frequent feedback cycles are essential in a dynamic environment (see Section 8.1) and can only really be provided through face-to-face communication.

Where teams are distributed geographically or even simply on different floors in a building it is important to implement a virtual collaboration/co-location environment, which is more than just video conferencing. The effective implementation of such an environment is the best, most suitable way to enable

good (face-to-face) communication.

The importance of face-to-face communication is also why most Agile frameworks define individual development team sizes of somewhere between 3 and 11 people. This is not a new idea: the majority of teams across all human endeavours, including sports teams, emergency service teams, military units or any other teams that deliver in dynamic environments mainly consist of 3 to 11 people.

1.2.2.7 Working software is the primary measure of progress (see Section 10.3)

While there are many excellent ways to measure progress and quality, the main objective of Agile is to deliver value, i.e. working software, to the customer in short time frames continuously. This makes the delivery of working software the primary measure of progress.

1.2.2.8 Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely (see Section 6.2)

Agile teams must work at a sustainable pace (see Section 8) to avoid people burning out, becoming ill or experiencing stress-related conditions.

Unrealistic and unsustainable time pressures may also cause corner cutting, which can lead to 'technical debt' in a product (see Section 10.4). Technical debt describes the long-term effects of ignoring or not recognising functional or technical problems in a system, and tends to result in systems that are full of defects, not effectively documented and badly designed. These systems are therefore difficult, expensive and sometimes even impossible to support, maintain and enhance.

There may also come a point at which personal motivation and commitment to the team is lost, potentially resulting in people leaving the team and possibly even the business. Such losses can cost the team and the organisation a huge amount of money, as significant investment will have been made in that person and in their efforts.

The majority of Agile frameworks include a role called the Agile lead (see Section 6.3) who is responsible for ensuring that the teams are operating at a sustainable pace. This is to ensure that functional quality is built into products, that technical debt is kept out of products and that people do not leave unexpectedly.

1.2.2.9 Continuous attention to technical excellence and good design enhances agility (see Section 10.4)

If a team does not apply technical excellence and good design from the start, there is a significant chance that major problems will only be identified late in the delivery lifecycle, at which point they can become extremely expensive to fix. Allowing hacks (quick fixes that are not designed or implemented at a fit-for-purpose level of quality) to occur, or inappropriate design or architectures to creep in leads to technical debt (see Section 10.4).

This means that it is important that the team develop and build the right products in

the right order in the optimum way. It is easy to deliver product stories without considering technical design holistically, but this will lead to a product that is fragile and unmaintainable; also, it will become increasingly difficult to develop product increments. If an area of a product is identified as being of a sub-standard design, the team should make sure that a 'refactoring' story (see Section 8.10) is raised to address this concern.

1.2.2.10 Simplicity – the art of maximising the amount of work not done – is essential (see Chapter 13)

Teams need to focus their efforts on developing a solution that is fit-for-purpose and only meets existing requirements. It is always tempting to build a product component that will meet current requirements, but is flexible enough to handle some perhaps-as-yet-undefined future requirement. However, this future requirement may never be needed or prioritised, which means the customer might end up with a product that is more expensive to maintain due to its complexity – which they didn't ask for or need.

It is equally important to concentrate on delivering value in the most effective way possible. This is where the concepts of Lean thinking (Liker, 2004) help to reduce waste and ensure the simplest 'value add' delivery chain possible is used (see Section 14.6).

1.2.2.11 The best architectures, requirements and designs emerge from self-organising teams (see Section 9.2)

With the right delivery environment and organisational culture a team can become self-organising (see Section 6.2), although this is not always a simple thing to achieve. For example, if detailed architectures, requirements and designs are defined without the team's input, there is the danger that the team will not buy into them, with the associated risk of loss of motivation. Agile assumes that teams typically know the best way forward at a detail level.

In more complex environments where many teams are working together to produce a product, it may be appropriate to produce high-level architectural and design principles. Teams can then align to these principles, but will be responsible for producing the detailed architecture and design, typically with support from the people who created the architecture and design principles.

1.2.2.12 At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly (see Section 11.2)

Agile deliveries follow an empirical process (i.e. a learning process – see Section 2.6). The three pillars of empirical processes are transparency, inspection and adaptation. Inspection and adaptation are of particular importance to this principle: at a frequent regular cadence, teams should take time to inspect their development process by reflecting on how things have progressed/developed since the last inspection and if there is anything that can be improved. Then these improvement ideas are used to adapt the development process.

This continuous improvement activity is known as a 'retrospective' (see Section

8.5). A retrospective will identify the areas and processes that work well, and those that need to be improved.